Penggunaan Algoritma Runut-Balik dalam Proses Resolusi *Query* dari Eksekusi Program dalam Bahasa Prolog

Edwin Rachman (NIM 13515042)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515042@std.stei.itb.ac.id

***Abstrak*—Eksekusi program dalam bahasa Prolog dilakukan menggunakan sebuah *expert system*, dimana *knowledge base*-nya adalah *source code* program dan *Inference engine* digunakan untuk melakukan resolusi *query*. *Inference engine* melakukan metode *backward chaining*, yaitu pengaplikasian aturan-aturan terhadap fakta-fakta yang ada, dari goal *query* hingga kebenaran atau ketidakbenaran dari *query* dapat dibuktikan. Jika *query* benar, maka *Inference engine* juga menyertakan instantiasi-instantiasi variabel yang perlu dilakukan agar dapat membuktikan kebenaran goal tersebut. Algoritma yang digunakan saat backward chaining adalah algoritma runut-balik.**

Kata kunci—backtracking; runut-balik; Prolog; logic; expert system; logic programming; query; resolution; inference engine; knowledge base

I. PENDAHULUAN

Dalam dunia pemrograman terdapat banyak sekali bahasa pemrograman, dan setiap bahasa pemrograman memiliki fitur, cara kerja, dan kegunaan mereka sendiri. Bahasa pemrograman dapat dikelompokkan berdasarkan paradigma pemrogramannya, yaitu pendekatan yang digunakan dalam bahasa pemrograman tersebut untuk memecahkan persoalan yang dihadapi.

Paradigma pemrograman yang paling banyak digunakan zaman sekarang adalah paradigma imperatif. Paradigma imperatif berarti bahwa urutan eksekusi program didefinisikan secara eksplisit dalam kode. Paradigma ini digunakan oleh hampir semua bahasa-bahasa yang populer sekarang seperti C, C++, Java, C#, dan Python. Tetapi, ternyata terdapat sebuah paradigma alternatif yang merupakan kebalikan total dari paradigma imperatif.

Paradigma tersebut adalah paradigma deklaratif, dimana kode sama sekali tidak mendefinisikan urutan eksekusi perintah-perintah dalam program. Kode program dalam bahasa berparadigma deklaratif menjelaskan komputasi apa yang harus dilakukan, bukan bagaimana komputasi tersebut harus dilakukan. Akan tetapi, menjalankan program yang hanya memiliki itu bukan hal yang mudah karena pada dasarnya komputer hanya memahami perintah-perintah dengan urutan yang eksplisit.

Paradigma deklaratif terbagi menjadi dua sub-paradigma untuk memecahkan masalah tersebut. Sub-paradigma yang pertama adalah functional programming, dan yang kedua adalah logic programming. Makalah ini akan fokus ke logic programming dan bahasa pemrograman Prolog yang menggunakan paradigma tersebut.

Logic programming mempergunakan sebuah *First Order Logic* dan *expert system* untuk menyelesaikan masalah yang dihadapkan pada program. *expert system* terdiri atas dua subsistem yaitu *knowledge base* dan *inference engine*. *Knowledge base* merupakan kumpulan fakta dan aturan mengenai suatu domain permasalahan. *Inference engine* adalah alat yang berfungsi untuk mengaplikasikan aturan-aturan dari *knowledge base* pada fakta-fakta yang sudah ada, sehingga menghasilkan fakta-fakta yang baru yang kemudian dapat diaplikasikan kembali.

Logic programming berusaha untuk melakukan penyelesaian masalah sebagai sebuah *query* terhadap program bahasa tersebut. Hal ini disebut sebagai *query resolution* atau resolusi *query*. Untuk melakukan resolusi *query*, *expert system* akan dijalankan dengan kode program sebagai *knowledge base* awalnya. Kemudian, *inference engine* akan berusaha untuk menghasilkan fakta-fakta yang sesuai sehingga dapat membuktikan kebenaran atau kesalahan dari *query* tersebut, dan jika benar, substitusi-substitusi apa yang perlu dilakukan pada variabel *query*nya. Resolusi *query* ini yang menjadi dasar dari berjalannya eksekusi program dengan paradigma logic programming.

Masalah baru sekarang adalah bagaimana cara *inference engine* menghasilkan fakta-fakta baru yang membuktikan goal *query* tersebut. Salah satu cara yang dapat ditempuh oleh *inference engine* adalah menggunakan metode yang disebut forward chaining.

Forward chaining adalah metode dimana dilakukan pengaplikasian aturan-aturan secara berulang pada setiap fakta yang mungkin dalam *knowledge base* sehingga menghasilkan fakta baru yang kemudian akan dimasukkan ke dalam *knowledge base*. Hal ini akan terus dilakukan hingga didapatkan fakta yang dapat membuktikan kebenaran dari goal *query*. Metode ini ternyata sangat tidak efisien karena memiliki

kompleksitas waktu dan tempat yang sangat besar. Ini dikarenakan metode forward chaining pada dasarnya adalah exhaustive search secara brute force.

Metode alternatif yang dapat dilakukan oleh *inference engine* adalah metode backward chaining, dimana pengaplikasian aturan terhadap fakta-fakta *knowledge base* dilakukan mundur dari goal *query*-nya. *Inference engine* akan mencari fakta yang sesuai dengan goal atau aturan yang mungkin dapat menghasilkan fakta yang sesuai dengan goal. Jika aturan yang telah diaplikasikan ternyata tidak dapat menghasilkan fakta yang sesuai, maka *inference engine* akan *backtrack* dan mencoba aplikasi aturan yang lain sehingga mencapai goal *query* atau dapat membuktikan bahwa goal *query* salah. Algoritma yang digunakan untuk metode backward chaining ini adalah algoritma *backtracking*

II. DASAR TEORI

A. Bahasa Pemrograman Prolog

Prolog adalah singkatan dari PROgramming in LOgic. Bahasa Prolog pertama kali dikembangkan oleh Alain Colmerauer dan P. Roussel di Universitas Maseilles Perancis pada tahun 1972. Prolog dikembangkan berdasarkan Kalkulus Predikat Order Pertama, atau *First Order Logic*.

Pemrograman dalam bahasa Prolog meliputi tiga hal utama, yaitu:

- Deklarasi sejumlah fakta mengenai suatu objek dan relasinya.
- Pendefinisian sejumlah rules/aturan mengenai suatu objek dan relasinya.
- Pertanyaan/*query* mengenai suatu objek dan relasinya.

Query adalah sebuah goal yang ingin diketahui kebenarannya, atau sebuah request ke *knowledge base* program Prolog. *Inference engine* Prolog akan berusaha untuk mendeduksi apakah *query* tersebut bernilai true atau false.

Clause/klausa adalah isi dari *knowledge base* sebuah program Prolog. Sebuah klausa terdiri dari fakta dan aturan

Fakta adalah predikat yang dianggap benar atau True dalam suatu program Prolog. Fakta dapat memiliki argumen (contoh: P(X, Y)).

Rules/aturan adalah sesuatu yang akan memberi tahu sistem prolog bagaimana mendeduksi fakta baru yang tidak terdaftar di database secara eksplisit. Aturan ditulis dalam bentuk implikasi (contoh: P :- Q, R, S. P bernilai True jika Q, R dan S bernilai True). Aturan juga dapat memiliki argumen.

Untuk melakukan eksekusi program, Prolog menggunakan sebuah *expert system* yang digunakan untuk melakukan resolusi *query* yang diberikan. [1]

B. *Expert system*

Expert system adalah sebuah program knowledge-based, yang dapat memberikan solusi berkualitas expert untuk sebuah masalah pada domain tertentu.

Sebuah *expert system* berusaha untuk mengemulasi metodologi dan performansi seorang expert manusia sebetulnya dalam penyelesaian masalah.

Fitur dari *expert system*:

- Terbuka untuk inspeksi (dapat memperlihatkan langkah-langkah antara dan menjawab pertanyaan mengenai proses dari solusinya)
- Mudah dimodifikasi (menambahkan dan mengurangi fakta atau aturan dari *knowledge base*)
- Heuristik (menggunakan *knowledge* untuk mendapatkan solusi)

Expert system terdiri atas dua subsistem yaitu *knowledge base* dan *inference engine*. *Knowledge base* merupakan kumpulan fakta dan aturan mengenai suatu domain permasalahan. *Inference engine* adalah alat yang berfungsi untuk mengaplikasikan aturan-aturan dari *knowledge base* pada fakta-fakta yang sudah ada, sehingga menghasilkan fakta-fakta yang baru yang kemudian dapat diaplikasikan kembali.

Masalah-masalah yang sesuai untuk solusi berbasis *expert system*:

- Keperluan solusi sesuai dengan biaya dan usaha untuk membangun sebuah *expert system*
- Keahlian manusia tidak tersedia di semua situasi dimana keahlian tersebut diperlukan
- Masalah dapat diselesaikan dengan teknik symbolic reasoning
- Domain permasalahan terstruktur dengan baik dan tidak memerlukan penalaran common sense
- Masalah tidak dapat diselesaikan menggunakan metode komputasi tradisional
- Expert yang kooperatif dan artikulat ada
- Masalah ukuran dan scope

Sistem produksi mengimplementasi sebuah perunutan graf secara DFS (*inference engine*). Subsistem tersebut akan mengamati perunutan graf tersebut untuk menjawab *query* user. [2]

C. Algoritma Runut-Balik

Backtracking atau runut-balik adalah algoritma yng berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik, yang merupakan perbaikan dari algoritma brute-force, secara sistematis mencari solusi persoalan di antar semua kemungkinan solusi yang ada. Dengan metodi ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja

yang selalu dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. Runut-balik lebih alami dinyatakan dalam algoritma rekursif.

Istilah runut-balik pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Selanjutnya, R.J. Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik dan penerapannya pada berbagai persoalan. Saat ini, algoritma runut-balik banyak diterapkan untuk program games (seperti permainan tic-tac-toe, menemukan jalan keluar dalam sebuah labirin, catur, dll) dan masalah-masalah pada bidang kecerdasan buatan (artificial intelligence).

Untuk menerapkan metode runut-balik, properti berikut didefinisikan:

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan n-tuple:
   $X = \{x_1, x_2, \ldots, x_n\}. x \in$ himpunan berhingga S_i
   Mungkin saja $S_1 = S_2 = \ldots = S_n$.
   Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1

2. Fungsi pembangkit nilai x_k
   Dinyatakan sebagai: T(k)
   T(k) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

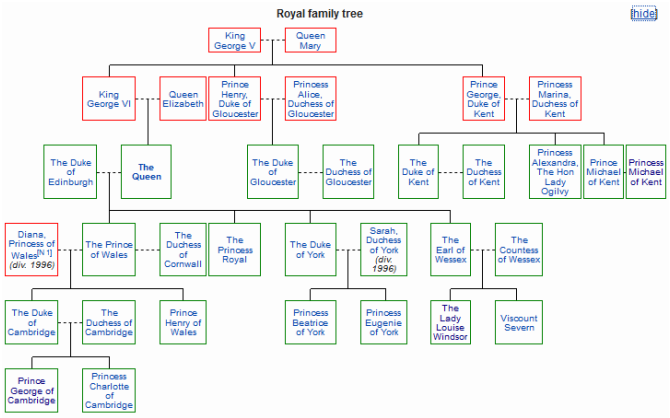
3. Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
   Dinyatakan sebagai: B(x_1, x_2, …, x_k)
   Fungsi pembatas menentukan apakah (x_1, x_2, …, x_k) mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k-1} dilanjutkan, tetapi jika tidak, maka (x_1, x_2, …, x_k) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.
   Fungsi pembatas tdiak selalu dinyatakan sebagai fungsi matematis. Ia dapat dinyatakan sebagai predikat yang bernilai true atau false, atau dalam bentuk lain yang ekivalen.

Dalam logic programming, solusi persoalannya adalah kebenaran atau ketidakbenaran goal *query* dari user, dan jika benar himpunan substitusi yang butuhkan untuk setiap variabel-variabel *query*. Fungsi pembangkitnya adalah aturan-aturan yang terdapat pada *knowledge base*. Fungsi pembatasnya adalah fungsi unification logika. [3]

Inference engine Prolog akan mengatur redo point untuk mencari seluruh solusi yang mungkin. Setiap langkah dan instantiasi variabel yang telah dilakukan akan dibalikkan ke state redo point tersebut jika eksekusi program mengharuskan terjadinya *backtracking*. Sebuah operatur khusus, yaitu operator cut (!) dimasukkan dalam program Prolog untuk meminta *inference engine* tidak melakukan *backtrack* ke bagian tertentu program.

III. Analisis dan Pembahasan

Keluarga royal Inggris merupakan keluarga yang sangat tua, berumur hingga ratusan tahun. Untuk setiap kasus yang akan dianalisis, domain permasalahan yang digunakan adalah silsilah keluarga royal Inggris seperti pada gambar di bawah ini.



Gambar 1. Silsilah Keluarga Royal Inggris

A. Kasus Pohon Keluarga Royal Inggris dengan *Query* Berbasis Fakta yang Berhasil False

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- meninggal(elizabeth_ii).
```

Gambar 2. Query Kasus A

Maksud dari *query* di atas adalah apakah Ratu Elizabeth II telah meninggal berdasarkan *knowledge base* domain masalah ini.

Terdapat sembilan fakta yang berhubungan dengan klausa `meninggal` yang terdapat pada *query*, yaitu:

```
meninggal(george_v).
meninggal(mary).
meninggal(george_vi).
meninggal(elizabeth).
meninggal(henry_duke_of_gloucester).
meninggal(alice).
meninggal(george_duke_of_kent).
meninggal(marina).
meninggal(diana).
```

Berikut adalah call trace dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
1    1  Call: meninggal(elizabeth_ii) ?
1    1  Fail: meninggal(elizabeth_ii) ?
```

Gambar 3. Call Trace Kasus A

Inference engine Prolog akan berusaha untuk melakukan proses unifikasi goal `meninggal(elizabeth_ii)` dengan setiap fakta di atas, seperti yang terlihat pada baris pertama call trace. Dapat dilihat bahwa tidak ada fakta yang dapat diunifikasi dengan goal tersebut. Goal dikatakan gagal, seperti yang terlihat pada baris kedua call trace. Oleh karena itu, *query* akan menghasilkan jawaban False, atau 'no' dalam bahasa Prolog.

```
(15 ms) no
```

Gambar 4. Hasil Resolusi Kasus A

B. Kasus Pohon Keluarga Royal Inggris dengan *Query* Berbasis Fakta yang Berhasil True

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- meninggal(diana).
```

Gambar 5. Query Kasus B

Maksud dari *query* di atas adalah apakah Putri Diana telah meninggal berdasarkan *knowledge base* domain masalah ini.

Seperti pada kasus sebelumnya, terdapat sembilan fakta yang berhubungan dengan klausa `meninggal` yang terdapat pada *query*.

Berikut adalah call trace dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
1    1  Call: meninggal(diana) ?
1    1  Exit: meninggal(diana) ?
```

Gambar 6. Call Trace Kasus B

Inference engine Prolog akan berusaha untuk melakukan proses unifikasi goal `meninggal(diana)` dengan setiap fakta yang telah disebutkan, seperti yang terlihat pada baris pertama call trace. Tidak seperti kasus sebelumnya, proses unifikasi ternyata berhasil karena goal *query* dapat diunifikasi dengan fakta `meninggal(diana).` Hal ini dapat dilihat pada baris kedua call trace. Oleh karena itu, *query* akan menghasilkan jawaban True, atau 'yes' dalam bahasa Prolog.

```
(16 ms) yes
```

Gambar 7. Hasil Resolusi Kasus B

C. Kasus Pohon Keluarga Royal Inggris dengan *Query* Berbasis Fakta yang Menggunakan Variabel

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- meninggal(X).
```

Gambar 8. Query Kasus C

Maksud dari *query* di atas adalah sebutkan semua X yang merupakan anggota keluarga royal Inggris yang sudah meninggal.

Seperti pada kasus sebelumnya, terdapat sembilan fakta yang berhubungan dengan klausa `meninggal` yang terdapat pada *query*.

Berikut adalah call trace awal dan hasil resolusi *query* pertama dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
1    1  Call: meninggal(_23) ?
1    1  Exit: meninggal(george_v) ?

X = george_v ? ;
```

Gambar 9. Call Trace Awal dan Hasil Resolusi Pertama Kasus C

Inference engine Prolog akan berusaha untuk melakukan proses unifikasi goal `meninggal(_23)` (catatan: variabel X diganti dengan variabel unik _23 oleh *inference engine*) dengan setiap fakta yang telah disebutkan, seperti yang terlihat pada baris pertama call trace. Fakta pertama yang akan diuji adalah `meninggal(george_v)`, yang ternyata berhasil diunifikasi dengan `meninggal(_23)`, dan variabel `_23` diinstantiasikan dengan konstanta `george_v` seperti yang terlihat pada baris kedua call trace dan hasil resolusi *query*nya.

Setelah ini, antarmuka konsol Prolog akan menanya ke user apakah *query* ini ingin dilanjutkan atau disudahi pada resolusi ini saja.

Berikut adalah call trace selanjutnya dan hasil resolusi kedua jika user memilih untuk melanjutkan *query*:

```
1    1  Redo: meninggal(george_v) ?
1    1  Exit: meninggal(mary) ?

X = mary ? ;
```

Gambar 10. Call Trace Kedua dan Hasil Resolusi Kedua Kasus C

Karena *inference engine* telah melakukan instantiasi variabel `_23` dengan konstanta `george_v`, maka *inference engine* akan melakukan proses runut-balik ke redo point sebelumnya, seperti yang terlihat pada baris pertama call trace. Kemudian *inference engine* akan berusaha untuk melakukan proses unifikasi goal `meninggal(_23)` dengan fakta selanjutnya, yaitu `meninggal(mary)`. Unifikasi tersebut ternyata berhasil, dan variabel `_23` diinstantiasikan dengan konstanta `mary` seperti yang terlihat pada baris kedua call trace dan hasil resolusi *query*nya.

Setelah ini, antarmuka konsol Prolog akan kembali menanya ke user apakah *query* ini ingin dilanjutkan atau disudahi pada resolusi ini saja. Proses ini akan berulang berkali-kali hingga setiap fakta dengan predikat `meninggal` telah diuji.

Berikut adalah call trace sisa dan hasil resolusi *query* ketiga hingga terakhir dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
      1    1  Redo: meninggal(mary) ?
      1    1  Exit: meninggal(george_vi) ?

X = george_vi ? ;
      1    1  Redo: meninggal(george_vi) ?
      1    1  Exit: meninggal(elizabeth) ?

X = elizabeth ? ;
      1    1  Redo: meninggal(elizabeth) ?
      1    1  Exit: meninggal(henry_duke_of_gloucester) ?

X = henry_duke_of_gloucester ? ;
      1    1  Redo: meninggal(henry_duke_of_gloucester) ?
      1    1  Exit: meninggal(alice) ?

X = alice ? ;
      1    1  Redo: meninggal(alice) ?
      1    1  Exit: meninggal(george_duke_of_kent) ?

X = george_duke_of_kent ? ;
      1    1  Redo: meninggal(george_duke_of_kent) ?
      1    1  Exit: meninggal(marina) ?

X = marina ? ;
      1    1  Redo: meninggal(marina) ?
      1    1  Exit: meninggal(diana) ?

X = diana

(47 ms) yes
```

Gambar 11. Call Trace Terakhir dan Hasil Resolusi Sisa Kasus C

Hasil resolusi *query* yang lengkap adalah True dengan variabel `X = george_v`, `mary`, `george_vi`, `elizabeth`, `henry_duke_of_gloucester`, `george_duke_of_kent`, `alice`, `marina`, atau `diana`.

D. Kasus Pohon Keluarga Royal Inggris dengan *Query* Disjungsi Dua Goal Berbasis Fakta yang Menggunakan Variabel

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- meninggal(X), perempuan(X).
```

Gambar 12. Query Kasus D

Maksud dari *query* di atas adalah sebutkan semua X yang merupakan anggota keluarga royal Inggris yang sudah meninggal, dan merupakan seorang perempuan.

Seperti pada kasus sebelumnya, terdapat sembilan fakta yang berhubungan dengan klausa `meninggal` yang terdapat pada *query*.

Terdapat 18 fakta yang berhubungan dengan klausa `perempuan` yang terdapat pada *query*:

```
perempuan(elizabeth).
perempuan(alice).
perempuan(marina).
perempuan(elizabeth_ii).
perempuan(birgitte).
perempuan(katharine).
perempuan(alexandra).
perempuan(princess_michael).
perempuan(diana).
perempuan(camilla).
perempuan(anne).
perempuan(sarah).
perempuan(sophie).
perempuan(catherine).
perempuan(beatrice).
perempuan(eugenie).
perempuan(louise).
perempuan(charlotte).
```

Berikut adalah call trace awal dan hasil resolusi *query* pertama dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
      1    1  Call: meninggal(_23) ?
      1    1  Exit: meninggal(george_v) ?
      2    1  Call: perempuan(george_v) ?
      2    1  Fail: perempuan(george_v) ?
      1    1  Redo: meninggal(george_v) ?
      1    1  Exit: meninggal(mary) ?
      2    1  Call: perempuan(mary) ?
      2    1  Exit: perempuan(mary) ?

X = mary ? ;
```

Gambar 13. Call Trace Awal dan Hasil Resolusi Pertama Kasus D

Tidak seperti tiga kasus sebelumnya, pada *query* ini terdapat dua goal, yaitu `meninggal(X)` dan `perempuan(X).` *Inference engine* akan berusaha untuk melakukan proses unifikasi goal pertama terlebih dahulu dengan setiap fakta yang telah disebutkan, seperti yang terlihat pada baris pertama call trace. Fakta pertama yang akan diuji adalah `meninggal(george_v)`, yang ternyata berhasil diunifikasi dengan `meninggal(_23)`, dan variabel `_23` diinstantiasikan dengan konstanta `george_v` seperti yang terlihat pada baris kedua call trace.

Kemudian *inference engine* akan berusaha untuk melakukan proses unifikasi goal kedua dengan `meninggal(george_v)` yang merupakan fakta hasil instantiasi sebelumnya, seperti yang terlihat pada baris ketiga call trace. Ternyata unifikasi tersebut tidak berhasil, seperti yang terlihat pada baris keempat call trace. Oleh karena itu, *inference engine* akan melakukan runut-balik ke redo point sebelumnya supaya instantiasi variabel `_23` dengan konstanta `george_v`, seperti yang terlihat pada baris kelima call trace.

Sekarang *inference engine* akan berusaha untuk melakukan proses unifikasi goal pertama dengan fakta kedua, yaitu `meninggal(mary).` Unifikasi tersebut ternyata berhasil, dan variabel `_23` diinstantiasikan dengan konstanta `mary` seperti yang terlihat pada baris kedua call keenam.

Kemudian *inference engine* akan berusaha untuk melakukan proses unifikasi goal kedua dengan `meninggal(mary)` yang merupakan fakta hasil instantiasi sebelumnya, seperti yang terlihat pada baris ketujuh call trace. Tidak seperti sebelumnya, unifikasi yang ini berhasil seperti yang terlihat pada baris kedelapan call trace.

Setelah ini, antarmuka konsol Prolog akan kembali menanya ke user apakah *query* ini ingin dilanjutkan atau disudahi pada resolusi ini saja. Proses ini akan berulang berkali-kali hingga setiap fakta dengan predikat `meninggal` telah diuji.

Berikut adalah call trace sisa dan hasil resolusi *query* ketiga hingga terakhir dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
        1    1  Redo: meninggal(mary) ?
        1    1  Exit: meninggal(george_vi) ?
        2    1  Call: perempuan(george_vi) ?
        2    1  Fail: perempuan(george_vi) ?
        1    1  Redo: meninggal(george_vi) ?
        1    1  Exit: meninggal(elizabeth) ?
        2    1  Call: perempuan(elizabeth) ?
        2    1  Exit: perempuan(elizabeth) ?

X = elizabeth ? ;
        1    1  Redo: meninggal(elizabeth) ?
        1    1  Exit: meninggal(henry_duke_of_gloucester) ?
        2    1  Call: perempuan(henry_duke_of_gloucester) ?
        2    1  Fail: perempuan(henry_duke_of_gloucester) ?
        1    1  Redo: meninggal(henry_duke_of_gloucester) ?
        1    1  Exit: meninggal(alice) ?
        2    1  Call: perempuan(alice) ?
        2    1  Exit: perempuan(alice) ?

X = alice ? ;
        1    1  Redo: meninggal(alice) ?
        1    1  Exit: meninggal(george_duke_of_kent) ?
        2    1  Call: perempuan(george_duke_of_kent) ?
        2    1  Fail: perempuan(george_duke_of_kent) ?
        1    1  Redo: meninggal(george_duke_of_kent) ?
        1    1  Exit: meninggal(marina) ?
        2    1  Call: perempuan(marina) ?
        2    1  Exit: perempuan(marina) ?

X = marina ? ;
        1    1  Redo: meninggal(marina) ?
        1    1  Exit: meninggal(diana) ?
        2    1  Call: perempuan(diana) ?
        2    1  Exit: perempuan(diana) ?

X = diana
```

Gambar 14. Call Trace Akhir dan Hasil Resolusi Sisa Kasus D

Hasil resolusi *query* yang lengkap adalah True dengan variabel `X` = `mary`, `elizabeth`, `alice`, `marina`, atau `diana`.

E. Kasus Pohon Keluarga Royal Inggris dengan *Query* Berbasis Aturan yang Berhasil False

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- saudara_dari(charles, diana).
```

Gambar 15. Query Kasus E

Maksud dari *query* di atas adalah apakah Pangeran Charles merupakan saudara kandung dari Putri Diana berdasarkan *knowledge base* domain masalah ini.

Tidak seperti kasus-kasus sebelumnya, tidak ada fakta yang berhubungan dengan klausa `saudara_dari` yang terdapat pada *query*. Akan tetapi, terdapat aturan klausa `saudara_dari`, yaitu: `saudara_dari(X, Y) :- orangtua_anak(Z, X), orangtua_anak(Z, Y), X \= Y`, yang berarti X adalah saudara kandung dari Y jika terdapat seorang Z yang merupakan orangtua dari X dan Y, dan X tidak sama dengan Y. Dari aturan ini, *inference engine* dapat membuat fakta-fakta baru yang mungkin sesuai dengan goal *query*.

Berikut adalah call trace dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
1    1  Call: saudara_dari(charles,diana) ?
2    2  Call: orangtua_anak(_86,charles) ?
2    2  Exit: orangtua_anak(philip,charles) ?
3    2  Call: orangtua_anak(philip,diana) ?
3    2  Fail: orangtua_anak(philip,diana) ?
2    2  Redo: orangtua_anak(philip,charles) ?
2    2  Exit: orangtua_anak(elizabeth_ii,charles) ?
3    2  Call: orangtua_anak(elizabeth_ii,diana) ?
3    2  Fail: orangtua_anak(elizabeth_ii,diana) ?
2    2  Redo: orangtua_anak(elizabeth_ii,charles) ?
2    2  Fail: orangtua_anak(_74,charles) ?
1    1  Fail: saudara_dari(charles,diana) ?
```

Gambar 16. Call Trace Kasus E

Inference engine pertama kali memanggil aturan `saudara_dari(X, Y)`, seperti yang terlihat pada baris pertama call trace. Supaya `saudara_dari(charles, diana)` menjadi benar, yang perlu dilakukan oleh *inference engine* adalah membuktikan setiap subgoalnya. Pertama, *inference engine* akan berusaha untuk melakukan proses unifkasi subgoal `orangtua_anak(Z, charles)` dengan fakta-fakta yang berhubungan, seperti yang terlihat pada baris kedua call trace.

Berikut adalah fakta-fakta yang berhubungan dengan `orangtua_anak(Z, charles)`:

```
orangtua_anak(philip, charles).

orangtua_anak(elizabeth_ii, charles).
```

Inference engine akan menguji fakta pertama `orangtua_anak(philip, charles)`, dan menginstantiasi variabel `Z` dengan konstanta `philip`, seperti yang terlihat pada baris ketiga call trace. Kemudian *inference engine* akan menguji fakta `orangtua_anak(philip, diana)`, yang tidak berhasil seperti yang terlihat pada baris kelima call trace. Oleh karena itu *inference engine* akan runut-balik kembali ke redo point sebelumnya. Hal yang sama juga dicoba dengan variabel `Z` diinstantiasi dengan `elizabeth_ii`, tetapi `orangtua_anak(elizabeth_ii, diana)` juga tidak berhasil.

Karena sudah tidak ada lagi fakta-fakta yang berhubungan dengan `orangtua_anak(Z, charles)` yang tersisa, maka `saudara_dari(charles, diana)` dinyatakan gagal oleh *inference engine*, seperti yang terlihat pada baris ke-12 call trace. Oleh karena itu, *query* akan menghasilkan jawaban False, atau 'no' dalam bahasa Prolog.

```
(15 ms) no
```

Gambar 17. Hasil Resolusi Kasus E

F. Kasus Pohon Keluarga Royal Inggris dengan *Query* Berbasis Aturan yang Berhasil True

Query yang merupakan goal dalam kasus ini adalah:

```
| ?- saudara_dari(henry, william).
```

Gambar 18. Query Kasus F

Maksud dari *query* di atas adalah apakah Pangeran Henry merupakan saudara kandung dari Pangeran William berdasarkan *knowledge base* domain masalah ini.

Kasus ini menggunakan klausa aturan `saudara_dari(X, Y)` seperti pada kasus sebelumnya.

Berikut adalah call trace dari eksekusi program Prolog ini dalam melakukan resolusi *query* di atas:

```
1    1  Call: saudara_dari(henry,william) ?
2    2  Call: orangtua_anak(_86,henry) ?
2    2  Exit: orangtua_anak(charles,henry) ?
3    2  Call: orangtua_anak(charles,william) ?
3    2  Exit: orangtua_anak(charles,william) ?
4    2  Call: henry\=william ?
5    3  Call: henry=william ?
5    3  Fail: henry=william ?
4    2  Exit: henry\=william ?
1    1  Exit: saudara_dari(henry,william) ?
```

Gambar 19. Call Trace Kasus F

Inference engine pertama kali memanggil aturan `saudara_dari(X, Y)`, seperti yang terlihat pada baris pertama call trace. Supaya `saudara_dari(henry, william)` menjadi benar, yang perlu dilakukan oleh *inference engine* adalah membuktikan setiap subgoalnya. Pertama, *inference engine* akan berusaha untuk melakukan proses unifkasi subgoal `orangtua_anak(Z, henry)` dengan fakta-fakta yang berhubungan, seperti yang terlihat pada baris kedua call trace.

Berikut adalah fakta-fakta yang berhubungan dengan `orangtua_anak(Z, henry)`:

```
orangtua_anak(charles, henry).

orangtua_anak(diana, henry).
```

Inference engine akan menguji fakta pertama `orangtua_anak(charles, henry)`, dan menginstantiasi variabel `Z` dengan konstanta `charles`, seperti yang terlihat pada baris ketiga call trace. Kemudian *inference engine* akan menguji fakta `orangtua_anak(charles, william)`, yang berhasil seperti yang terlihat pada baris kelima call trace. Karena dua subgoal pertama sudah benar, maka akan diuji subgoal ketiga, yaitu: `henry \= william`, yang berhasil jika `henry = william` gagal, seperti yang terlihat pada baris ke-8 dan ke-9 call trace.

Karena semua subgoal dari `saudara_dari(henry, william)` benar, maka klausa tersebut juga benar. Oleh karena itu, *query* akan menghasilkan jawaban True, atau 'yes' dalam bahasa Prolog.

```
true ?

(16 ms) yes
```

Gambar 20. Hasil Resolusi Kasus F

IV. KESIMPULAN DAN SARAN

A. Kesimpulan

Kesimpulan yang dapat diperoleh dari penulisan makalah "Penggunaan Algoritma Runut-Balik dalam Proses Resolusi *Query* dari Eksekusi Program dalam Bahasa Prolog" ini berdasarkan uraian yang telah dijelaskan pada ketiga bab sebelumnya adalah sebagai berikut:

- Algoritma runut-balik dapat digunakan sebagai dasar dari operasi *inference engine* metode backward chaining dalam eksekusi sebuah program Prolog.
- Algoritma runut-balik yang digunakan sebagai dasar dari backward chaining lebih efisien dibandingkan dengan algoritma brute force yang digunakan sebagai dasar dari forward chaining, karena memiliki kompleksitas waktu dan tempat yang lebih kecil.
- Proses resolusi *query* yang dilakukan oleh *inference engine* menggunakan algoritma runut-balik dapat digunakan untuk mengeksekusi program dengan kode deklaratif dalam bahasa Prolog.

B. Saran

Dalam penulisan makalah ini, terdapat beberapa saran-saran yang diperoleh untuk perkembangan yang lebih lanjut bidang *logic programming* dan paradigm pemrograman deklaratif:

- Disarankan kepada para programmer, expert, dan peneliti dalam bidang informatika agar tidak menganggap remeh paradigma-paradigma lain yang tidak popular, karena setiap paradigma pemrograman memiliki fungsimya tersendiri.
- Disarankan kepada para peneliti bidang pembelajaran mesin, NLP, dan inteligensi buatan untuk lebih memperhatikan *logic programming* karena memiliki potensi yang sangat besar dalam bidang-bidang tersebut

UCAPAN TERIMA KASIH

Saya mengucapkan terima kasih kepada dosen pengampu mata kuliah IF2211 – Strategi Algoritma, bapak Dr. Ir. Rinaldi Munir, M.T., ibu Masayu Leyla Khodra S.T., M.T., dan ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc, karena telah mengajar dan membimbing kami selama satu semester ini. Saya juga mengucapkan terima kasih kepada teman-teman seangkatan yang secara langsung dan tidak langsung membantu saya dalam proses pembelajaran saya dalam mata kuliah ini.

DAFTAR PUSTAKA

[1] Maulidevi, Nur Ulfa. 2016. *Introduction to Prolog*. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

[2] Maulidevi, Nur Ulfa. 2016. *Expert Systems*. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

[3] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Mei 2017

Edwin Rachman
NIM 13515042

LAMPIRAN

A. *Source Code* Lengkap Program Prolog Silsilah Keluarga Royal Inggris

```
%FAKTA-FAKTA
%generasi pertama
laki_laki(george_v).

%generasi kedua
laki_laki(george_vi).
laki_laki(henry_duke_of_gloucester).
laki_laki(george_duke_of_kent).

%generasi ketiga
laki_laki(philip).
laki_laki(richard).
laki_laki(edward_duke_of_kent).
laki_laki(prince_michael).

%generasi keempat
laki_laki(charles).
laki_laki(andrew).
laki_laki(edward).

%generasi kelima
laki_laki(william).
laki_laki(henry).
laki_laki(james).

%generasi keenam
laki_laki(george).

%generasi pertama
perempuan(mary).

%generasi kedua
perempuan(elizabeth).
perempuan(alice).
perempuan(marina).

%generasi ketiga
perempuan(elizabeth_ii).
perempuan(birgitte).
perempuan(katharine).
perempuan(alexandra).
perempuan(princess_michael).

%generasi keempat
perempuan(diana).
perempuan(camilla).
perempuan(anne).
perempuan(sarah).
perempuan(sophie).

%generasi kelima
perempuan(catherine).
perempuan(beatrice).
perempuan(eugenie).
perempuan(louise).

%generasi keenam
perempuan(charlotte).

%generasi pertama
suami_istri(george_v, mary).

%generasi kedua
suami_istri(george_vi, elizabeth).
suami_istri(henry_duke_of_gloucester,
alice).
suami_istri(george_duke_of_kent, marina).

%generasi ketiga
suami_istri(philip, elizabeth_ii).
suami_istri(richard, birgitte).
suami_istri(edward_duke_of_kent,
katharine).
suami_istri(prince_michael,
princess_michael).

%generasi keempat
suami_istri(charles, camilla).
suami_istri(edward, sophie).

%generasi kelima
suami_istri(william, catherine).

%generasi pertama
orangtua_anak(george_v, george_vi).
orangtua_anak(george_v,
henry_duke_of_gloucester).
orangtua_anak(george_v,
george_duke_of_kent).
orangtua_anak(mary, george_vi).
```

```
orangtua_anak(mary,
henry_duke_of_gloucester).
orangtua_anak(mary, george_duke_of_kent).

%generasi kedua
orangtua_anak(george_vi, elizabeth_ii).
orangtua_anak(elizabeth, elizabeth_ii).
orangtua_anak(henry_duke_of_gloucester,
richard).
orangtua_anak(alice, richard).
orangtua_anak(george_duke_of_kent,
edward_duke_of_kent).
orangtua_anak(george_duke_of_kent,
alexandra).
orangtua_anak(george_duke_of_kent,
prince_michael).
orangtua_anak(marina,
edward_duke_of_kent).
orangtua_anak(marina, alexandra).
orangtua_anak(marina, prince_michael).

%generasi ketiga
orangtua_anak(philip, charles).
orangtua_anak(philip, anne).
orangtua_anak(philip, andrew).
orangtua_anak(philip, edward).
orangtua_anak(elizabeth_ii, charles).
orangtua_anak(elizabeth_ii, anne).
orangtua_anak(elizabeth_ii, andrew).
orangtua_anak(elizabeth_ii, edward).

%generasi keempat
orangtua_anak(charles, william).
orangtua_anak(charles, henry).
orangtua_anak(diana, william).
orangtua_anak(diana, henry).
orangtua_anak(andrew, beatrice).
orangtua_anak(andrew, eugenie).
orangtua_anak(sarah, beatrice).
orangtua_anak(sarah, eugenie).
orangtua_anak(edward, louise).
orangtua_anak(edward, james).
orangtua_anak(sophie, louise).
orangtua_anak(sophie, james).

%generasi kelima
orangtua_anak(william, george).
orangtua_anak(william, charlotte).
orangtua_anak(catherine, george).
orangtua_anak(catherine, charlotte).

%generasi pertama
meninggal(george_v).
meninggal(mary).

%generasi kedua
meninggal(george_vi).
meninggal(elizabeth).
meninggal(henry_duke_of_gloucester).
meninggal(alice).
meninggal(george_duke_of_kent).
meninggal(marina).

%generasi keempat
meninggal(diana).

%ATURAN-ATURAN
nikah_dengan(X, Y) :- suami_istri(X, Y).
nikah_dengan(X, Y) :- suami_istri(Y, X).

saudara_dari(X, Y) :- orangtua_anak(Z,
X), orangtua_anak(Z, Y), X \= Y.

sepupu_dari(X, Y) :- orangtua_anak(A, X),
orangtua_anak(B, Y), saudara_dari(A, B).

anak_tunggal(X) :- orangtua_anak(Z, X),
\+ (orangtua_anak(Z, Y), X \= Y).

keturunan_dari(X, Y) :- orangtua_anak(Y,
X).
keturunan_dari(X, Y) :- orangtua_anak(Z,
X), keturunan_dari(Z, Y).
```

B. Gambar Silsilah Keluarga Royal Inggris